python for Computational Problem Solving - pCPS - Data and Expressions Lecture Slides - Class #5 to Class#6

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

python for Computational Problem Solving Syllabus

Unit I: Computational Problem Solving - 12 Hours

Limits of Computational Problem Solving - Computer Algorithm - Computer Hardware - Digital Computer - Operating System- Limits of IC technology - Computer Software - Syntax, semantics and program translation ,Introduction to Python Programming Language, IDLE Python Development Environment, Output function - variables, types and id,input function , operators and expressions, Control structures .

T1: 1.1 – 1.7
T1: 2.1 - 2.4
T1: 3.1 – 3.4

- 2 Data and Expressions
  - MOTIVATION
  - FUNDAMENTAL CONCEPTS
  - 2.1 Literals
  - 2.2 Variables and Identifiers
  - 2.3 Operators
  - 2.4 Expressions and Data Types

pCPS 2.1.1 Literals

- A **literal** is a sequence of one of more characters that stands for itself, such as the literal 12, “PESU”, 12.5, 12+14j etc

pCPS 2.1.2 Numeric Literals

- A **numeric literal** is a literal containing only the digits 0–9, an optional sign character (1 or 2), and a possible decimal point.
- If a numeric literal contains a decimal point, then it denotes a floating-point value , or “ float ”
- Commas are never used in numeric literals.
- Numeric literals without a provided sign character denote positive values, an explicit positive sign character is rarely used.

Numeric Literals						
integer values	floating-point values					incorrect
5	5.	5.0	5.125	0.0005	5000.125	5,000.125
2500	2500.	2500.0	2500.125			2,500 2,500.125
+2500	+2500.	+2500.0	+2500.125			+2,500 +2,500.125
-2500	-2500.	-2500.0	-2500.125			-2,500 -2,500.125

pCPS 2.1.2 Numeric Literals

- There is **no limit** to the size of an integer that can be represented in python

- Limits of Range in Floating-Point Representation
  - Floating-point values, have both a limited range and a limited precision .

  - **python** uses a **double-precision** standard format (**IEEE 754**) providing a range of 10^{-308} to 10^{308} with 16 to 17 digits of precision

  - It is important to understand the **limitations** of **floating-point** representation.

  - **Arithmetic Overflow** , is a condition that occurs when a calculated result is too large in magnitude (size) to be represented

  - **Arithmetic Underflow** , is a condition that occurs when a calculated result is too small in magnitude to be represented

pCPS 2.1.2 Numeric Literals

```
# PES University
# Course - python for Computational Problem Solving - (pCPS)
# Nitin V Pujari
```

```
# python numeric literals
print(10)
print(+1000)
print(-1000)
print(0.25)
print(2.25)
print(3.1415932)
```

```
10
1000
-1000
0.25
2.25
3.1415932
```

```
type((1,132))
```

```
tuple
```

```
1,192,37
```

```
(1, 192, 37)
```

pCPS 2.1.2 Numeric Literals

```
import sys
print(sys.int_info)
print(sys.maxsize)
```

```
sys.int_info(bits_per_digit=30, sizeof_digit=4)
9223372036854775807
```

```
print(sys.float_info)
```

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-102
1, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

```
print(sys.float_repr_style)
```

```
short
```

pCPS 2.1.2 Numeric Literals

```
# Arithmetic Overflow , is a condition that occurs when a calculated result is too large in magnitude (size)
# to be represented
print(1.5e200 * 2.0e210)
print(type(1.5e200 * 2.0e210))
```

```
inf
<class 'float'>
```

```
# Arithmetic Overflow , is a condition that occurs when a calculated result is too large in magnitude (size)
# to be represented
print(type(1.7976931348623157e+308 ** 2))
print(1.7976931348623157e+308 ** 2)
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
/tmp/ipykernel_5643/540608372.py in <module>
      1 # Arithmetic Overflow , is a condition that occurs when a calculated result is too large in magnitude (size)
      2 # to be represented
----> 3 print(type(1.7976931348623157e+308 ** 2))
      4 print(1.7976931348623157e+308 ** 2)

OverflowError: (34, 'Numerical result out of range')
```

pCPS 2.1.2 Numeric Literals

```
# Arithmetic Underflow , is a condition that occurs when a calculated result is
# too small in magnitude to be represented
print(1.0e2300 / 1.0e100)
print(type(1.0e2300 / 1.0e100))
```

```
inf
<class 'float'>
```

```
print(1.2e200 * 2.4e100)
```

```
2.88e+300
```

```
print(1.2e200 / 2.4e100)
```

```
5e+99
```

```
print(1.2e200 * 2.4e200)
```

```
inf
```

```
print(1.2e2200 / 2.4e200)
```

```
inf
```

pCPS 2.1.2 Numeric Literals

- **Arithmetic** **overflow** and **arithmetic** **underflow** are relatively easily detected.
- The **loss** of **precision** that can result in a calculated result, however, is an issue of concern
- Any floating-point representation necessarily contains only a finite number of digits, what is stored for many floating-point values is only an approximation of the true value
- No matter how **python** chooses to **display** calculated **results**, the value stored is **limited** in both the **range** of **numbers** that can be represented and the **degree** of **precision**.
- For **most** everyday **applications**, **this** slight **loss** in **accuracy** is of no practical concern.
- In scientific computing and other applications in which precise calculations are required, this is something that the programmer must be keenly aware of

pCPS 2.1.2 Numeric Literals

```
# Any floating-point representation necessarily contains only a finite number of digits,
# what is stored for many floating-point values is only an approximation of the true value
print(1/3)
print(1/3+1/3)
print(1/3+1/3+1/3)
print(1/3+1/3+1/3+1/3+1/3+1/3)
print(6 *(1/3))
```

```
0.33333333333333333
0.66666666666666666
1.0
1.9999999999999998
2.0
```

pCPS 2.1.2 Numeric Literals

```
print(1/10)
print(1/10+1/10)
print(1/10+1/10+1/10)
print(1/10+1/10+1/10+1/10+1/10+1/10)
print(6 *(1/10))
print(6 *1/10)
```

```
0.1
0.2
0.30000000000000004
0.6
0.6000000000000001
0.6
```

pCPS 2.1.2 Numeric Literals

- **Floating-point** values **may** contain an **arbitrary** number of **decimal** places, the built-in **format** function can be used to produce a numeric string version of the value containing a specific number of decimal places

- Format specifier '.2f' rounds the result to two decimal places of accuracy in the string produced.

- For very large (or very small) values 'e' can be used as a format specifier

- A **comma** in the format specifier **adds** **comma** separators to the result

pCPS 2.1.2 Numeric Literals

```
#PES University
# Course - pCPS
# Nitin V Pujari
```

```
12/5
```

```
2.4
```

```
5/7
```

```
0.7142857142857143
```

```
format(12/5, '.2f')
```

```
'2.40'
```

```
format(5/7, '.2f')
```

```
'0.71'
```

```
format(2 ** 100, '.6e')
```

```
'1.267651e+30'
```

```
format(13402.25, ',.2f')
```

```
'13,402.25'
```

```
format(11/12, '.2f')
```

```
'0.92'
```

```
format(11/12, '.2e')
```

```
'9.17e-01'
```

```
format(11/12, '.3f')
```

```
'0.917'
```

```
format(11/12, '.3e')
```

```
'9.167e-01'
```

pCPS 2.1.3 String Literals

- **String** literals , or “ strings ,” represent a **sequence** of **characters**
- In Python, string literals may be delimited (surrounded) by a matching pair of either single (') or double (") quotes.
- Strings must be contained all on one line , except when delimited by triple quotes
- A string may contain zero or more characters, including letters, digits, special characters, and blanks
- A string consisting of only a pair of matching quotes (with nothing in between) is called the empty string , which is different from a string containing only blank characters.
- Strings may also contain quote characters as long as different quotes are used to delimit the string
- If the string containing **python’s** were delimited with single quotes, the apostrophe (single quote) would be considered the matching closing quote of the opening quote, leaving the last final quote unmatched

pCPS 2.1.3 String Literals

- Python allows the use of more than one type of quote for such situations.
- In this course, the convention that will be used is single quotes for delimiting strings, and only use double quotes when needed.

```
#PES University
# Course - pCPS
# Nitin V Pujari
```

```
# In Python, string literals may be delimited (surrounded) by a matching pair of either single (')
# or double (") quotes.

# Strings must be contained all on one line , except when delimited by triple quotes
print('PES University')
print("PES University")
print("""PES University
      """)
```

```
PES University
PES University
PES University
```

pCPS 2.1.3 String Literals

```
# A string may contain zero or more characters, including letters, digits, special characters, and blanks

# A string consisting of only a pair of matching quotes (with nothing in between) is called the empty string ,
# which is different from a string containing only blank characters.

print('PES University B.Tech 1 Semester 2021')
print(len('PES University B.Tech 1 Semester 2021'))
print(len(''))
print(len(' '))

# print('pCPS',''s'', Laboratory') #results in a syntax error
print("PCPS's Laboratory")
```

```
PES University B.Tech 1 Semester 2021
37
0
1
PCPS's Laboratory
```

pCPS 2.1.3 String Literals

```
print('Hello')
```

```
Hello
```

```
print('Hello")
```

```
  File "/tmp/ipykernel_15151/210020672.py", line 1
    print('Hello")
                  ^
SyntaxError: EOL while scanning string literal
```

```
print('Let's Go')
```

```
  File "/tmp/ipykernel_15151/1282651989.py", line 1
    print('Let's Go')
              ^
SyntaxError: invalid syntax
```

```
print("Hello")
```

```
Hello
```

```
print("Let's Go!')
```

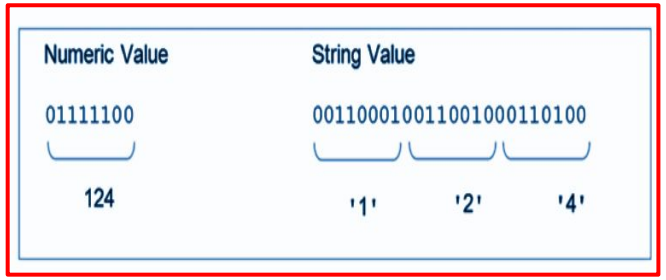
```
  File "/tmp/ipykernel_15151/3421626062.py", line 1
    print("Let's Go!')
                      ^
SyntaxError: EOL while scanning string literal
```

```
print("Let's go!")
```

```
Let's go!
```

pCPS 2.1.3 String Literals

- **Everyone** in the **world** should be able to **use** their **own** language on **phones** and **computers**
- There needs to be a way to encode (represent) characters within a computer.
- **Unicode** encoding scheme is intended to be a universal encoding scheme.
- **Unicode** is actually a collection of different encoding schemes utilizing between 8 and 32 bits for each character.
- **UTF** means **U**niversal **T**ransformation **F**ormat (https://home.unicode.org/)
- The default encoding in python uses UTF-8 , an 8-bit encoding compatible with ASCII, an older, still widely used encoding scheme
- Currently, there are over 100,000 Unicode-defined characters for many of the world
- Unicode is capable of defining more than 4 billion characters
- UTF-8 encoded characters have an ordering with sequential numerical values.



pCPS 2.1.3 String Literals

- There needs to be a way to encode (represent) characters within a computer.
- **Unicode** encoding scheme is intended to be a universal encoding scheme.
- Unicode is actually a collection of different encoding schemes utilizing between **8** and **32** **bits** for each character.
- The default encoding in python uses **UTF-8** , an **8-bit encoding** compatible with ASCII, an older, still widely used encoding scheme
- Currently, there are over **100,000 Unicode-defined characters** for many of the languages around the **world**
- **Unicode** is **capable** of defining more than **4 billion characters**
- UTF-8 encoded characters have an ordering with sequential numerical values.

```
#PES University
# Course - pCPS
# Nitin V Pujari
```

```
print('A')
print(chr(65))
print(ord('A'))
print(1)
print(ord('1'))
print(chr(1))
```

```
A
A
65
1
49
```

```
ord('1')
```

```
49
```

```
ord('1')
```

```
49
```

```
ord('2')
```

```
50
```

```
chr(97)
```

```
'a'
```

```
chr(90)
```

```
'Z'
```

pCPS 2.1.4 Control Characters

- Control characters are special characters that are not displayed on the screen.
- They control the display of output in addition to other things.
- Control characters do not have a corresponding keyboard character.
- They are represented by a combination of characters called an **escape sequence** .
- An **escape sequence** begins with an escape character that causes the sequence of characters following it to "escape" their normal meaning.
- The backslash (\) serves as the escape character in python.

```
#PES University
# Course - pCPS
# Nitin V Pujari
```

```
print('PES \nUniversity')
```

```
PES
University
```

```
print(len('\n'))
```

```
1
```

```
print(ord('\n'))
```

```
10
```

```
print('PES chr(10)University')
```

```
PES chr(10)University
```

```
print('PES',chr(10),'University')
```

```
PES
 University
```

```
print('Hello World')
```

```
Hello World
```

```
print('Hello\nWorld')
```

```
Hello
World
```

```
print('Hello World\n')
```

```
Hello World
```

```
print('Hello\n\nWorld')
```

```
Hello

World
```

```
print('Hello World\n\n')
```

```
Hello World
```

```
print(1, '\n', 2, '\n', 3)
```

```
1
 2
 3
```

```
print('\nHello World')
```

```

Hello World
```

```
print('\n', 1, '\n', 2, '\n', 3)
```

```

 1
 2
 3
```

pCPS 2.1.5 String Formatting

- We saw earlier the use of built-in function format for controlling how numerical values are displayed.
- Let us look at how the format function can be used to control how strings are displayed
- As given above, the format function has the form,
  - format(value, format_specifi er)
  - **value** is the value to be displayed, and **format_specifier** can contain a combination of formatting options
- Formatted strings are left-justified by default.

```
#PES University
# Course - pCPS
# Nitin V Pujari
```

```
print('PES \nUniversity')
```

```
PES
University
```

```
print(len('\n'))
```

```
1
```

```
print(ord('\n'))
```

```
10
```

```
print('PES chr(10)University')
```

```
PES chr(10)University
```

```
print('PES',chr(10),'University')
```

```
PES
 University
```

```
print('Hello World')
```

```
Hello World
```

```
print('Hello\nWorld')
```

```
Hello
World
```

```
print('Hello World\n')
```

```
Hello World

```

```
print('Hello\n\nWorld')
```

```
Hello

World
```

```
print('Hello World\n\n')
```

```
Hello World


```

```
print(1, '\n', 2, '\n', 3)
```

```
1
 2
 3
```

```
print('\nHello World')
```

```

Hello World
```

```
print('\n', 1, '\n', 2, '\n', 3)
```

```

 1
 2
 3
```

pCPS 2.1.6 Implicit and Explicit Joining

- A program line may be too long to fit in the Python-recommended maximum length of 79 characters.

- **Implicit Line Joining**

  - There are certain delimiting characters that allow a logical program line to span more than one physical line.

  - This includes matching **parentheses**, **square brackets**, **curly braces**, and **triple quotes**.

  - Matching quotes (except for triple quotes) must be on the same physical line

pCPS 2.1.6 Implicit and Explicit Joining

- A program line may be too long to fit in the Python-recommended maximum length of 79 characters.

- **Explicit Line Joining**

  - program lines may be explicitly joined by use of the backslash (\) character.

  - Program lines that end with a backslash that are not part of a literal string (that is, within quotes) continue on the following line

THANK YOU - End of Class#6

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University
nitin.pujari@pes.edu

For Course Digital Deliverables visit www.pesuacademy.com

pCPS 2.2.1 Variables and Identifiers

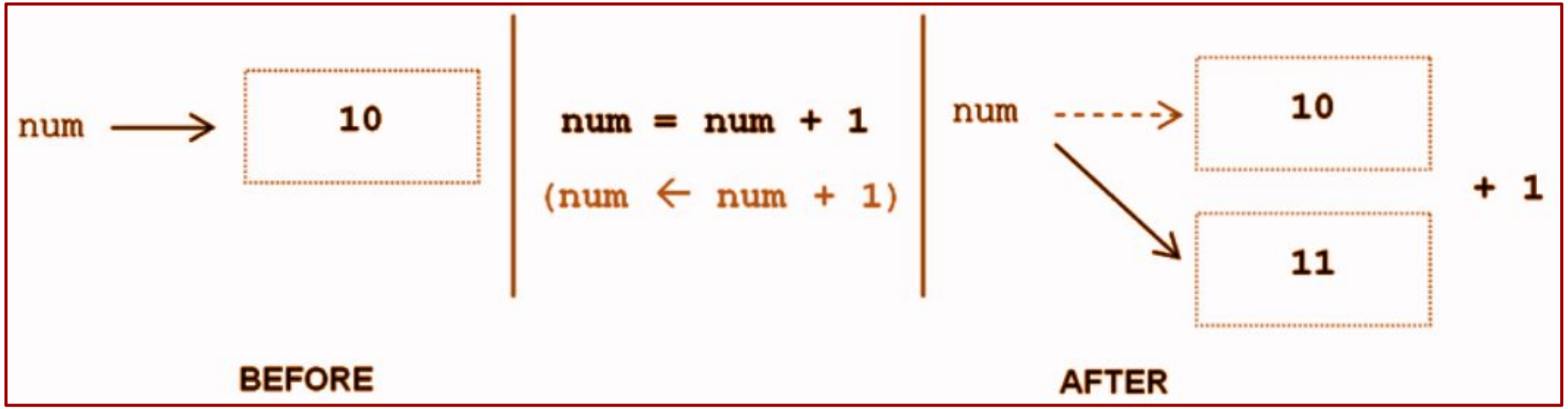
- The true usefulness of a computer program is the ability to operate on different values each time the program is executed.
- This is provided by the notion of a **variable** .
- A **variable** is a name (identifier) that is associated with a value
- A **variable** can be **assigned** different values during a program's execution—hence, the name "variable."
- Wherever a **variable** appears in a program (except on the **left-hand side** of an assignment statement), it is the **value** associated with the variable that is used , and not the variable's name
- Variables are assigned values by use of the **assignment** operator , =

pCPS 2.2.1 Variables and Identifiers

- **Assignment** statements often look wrong to novice programmers. **Mathematically**, num = num + 1 does **not** make **sense**.

- In **computing**, however, it is used to **increment** the value of a **given** **variable** by one.

- It is more appropriate, therefore, to **think** of the **=** symbol as an **arrow(⟵)** symbol

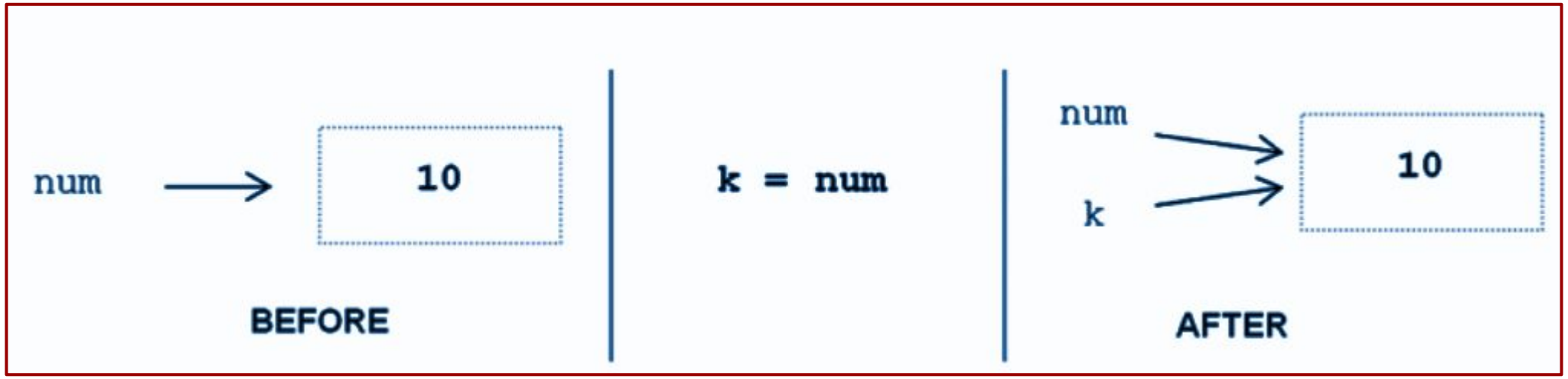


pCPS 2.2.1 Variables and Identifiers

- When thought of this way, it makes clear that the right side of an assignment is evaluated first, then the result is assigned to the variable on the left .

- An arrow symbol is not used simply because there is no such character on a standard computer keyboard.

- Variables may also be assigned to the value of another variable

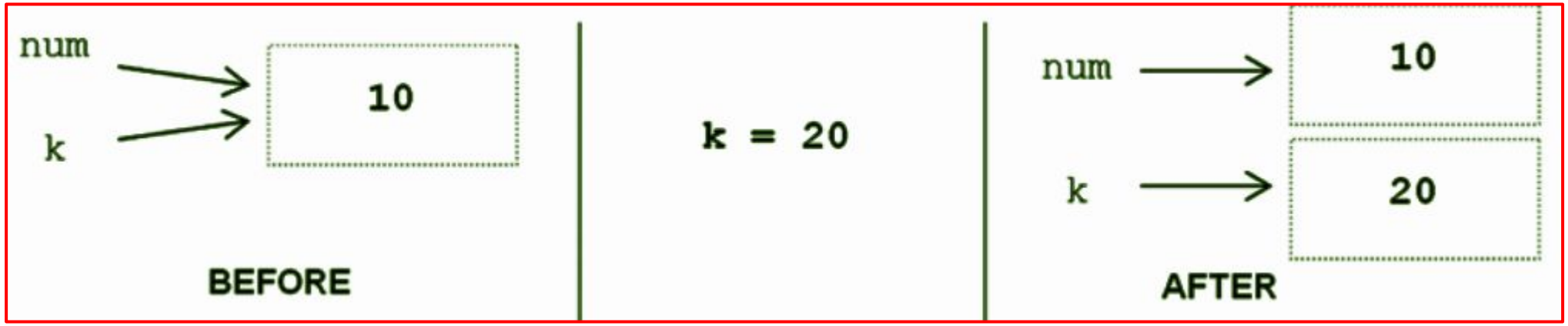


pCPS 2.2.1 Variables and Identifiers

- Variables num and k are both associated with the same literal value 10 in memory.
- One way to see this is by use of built-in function id
- The id function produces a unique number identifying a specific value (object) in memory.
- Since variables are meant to be distinct, it would appear that this sharing of values would cause problems.

pCPS 2.2.1 Variables and Identifiers

- If the value of num changed, would variable k change along with it ?
- This cannot happen in this case because the variables refer to integer values, and integer values are immutable .
- An immutable value is a value that cannot be changed.
- Thus, both will continue to refer to the same value until one (or both) of them is reassigned



pCPS 2.2.1 Variables and Identifiers

- If no other variable references the memory location of the original value, the memory location is deallocated (that is, it is made available for reuse).

- In python the same variable can be associated with values of different type during program execution

```
i=10
j=i
print(id(i))
print(id(j))

i=400

print(id(i))
print(id(j))

i=j

print(id(i))
print(id(j))
```

```
94379681423104
94379681423104
140094255235504
94379681423104
94379681423104
94379681423104
```

pCPS 2.2.2 Variables and Keyboard Input

- The **value** that is assigned to a given **variable** **need** **not** have to be specified in the program
- The **value** can come from the user by use of the input function
- As shown on the side, the variable name is assigned the string **'PESU'**.
- If the **user** **hit** **return** without entering any value, name would be assigned to the **empty** **string** ('')
- All input is returned by the input function as a string type.
- For the input of numeric values, the response must be converted to the appropriate type.

```
Name = input('What is the name of your University ?')
print(Name)
print(type(Name))
```

```
What is the name of your University ?PESU
PESU
<class 'str'>
```

```
Name = input('What is the name of your University ?')
print(Name)
print(type(Name))
```

```
What is the name of your University ?

<class 'str'>
```

pCPS 2.2.2 Variables and Keyboard Input

- For the input of numeric values, the response must be converted to the appropriate type.
- python provides built-in type conversion functions int () and float () for this purpose

```
Credits = input('How many credits you have this semester? ')
print('Curently', type(Credits))
Credits = int(Credits)
print('Now', type(Credits))
print('Credits',Credits)
```

```
How many credits you have this semester? 23
Curently <class 'str'>
Now <class 'int'>
Credits 23
```

```
SGPA = input('Predicted SGPA? ')
print('Curently', type(SGPA))
SGPA = float(SGPA)
print('Now', type(SGPA))
print('SGPA',SGPA)
```

```
Predicted SGPA? 8.35
Curently <class 'str'>
Now <class 'float'>
SGPA 8.35
```

pCPS 2.2.2 Variables and Keyboard Input

- Note that the program lines above could be combined as follows

```
Credits = int(input('How many credits you have this semester? '))
print('Now', type(Credits))
print('Credits',Credits)

SGPA = float(input('Predicted SGPA? '))
print('Now', type(SGPA))
print('SGPA',SGPA)
```

```
How many credits you have this semester? 23
Now <class 'int'>
Credits 23
Predicted SGPA? 9.41
Now <class 'float'>
SGPA 9.41
```

pCPS 2.2.3 Identifiers in python

- An **identifier** is a sequence of one or more characters used to provide a name for a given program element.
- Variable names line, Credits, and SGPA are each identifiers.
- python is **Case Sensitive** , thus, Credits is different from credits
- **Identifiers** may contain **letters** and **digits**, but **cannot begin** with a **digit**.
- The underscore character,**_**, is also allowed to aid in the readability of long identifier names.
- **_ (underscore)** should not be used as the first character, however, as identifiers beginning with an underscore have special meaning in python
- Spaces are not allowed as part of an **identifier**.
- This is a common error since some operating systems allow spaces within file names.
- Any **identifier** containing a space character would be considered two separate **identifiers**

pCPS 2.2.3 Identifiers in python

```python
print('Valid Identifiers')
One = 1
Two = 2.0
Three = 'PESU'
Four2Complex = 1+1j
Semester_Grade_Point_Average = 9.41
print(One)
print(Two)
print(Three)
print(Four2Complex)
print(Semester_Grade_Point_Average)
```

```
Valid Identifiers
1
2.0
PESU
(1+1j)
9.41
```

```python
print('InValid Identifiers')
'One' = 1
```

```
  File "/tmp/ipykernel_5446/1001126093.py", line 2
    'One' = 1
    ^
SyntaxError: cannot assign to literal
```

```python
print('InValid Identifiers')
Four 2 Complex = 1+1j
```

```
  File "/tmp/ipykernel_5446/1681098473.py", line 2
    Four 2 Complex = 1+1j
         ^
SyntaxError: invalid syntax
```

```python
print('InValid Identifiers')
2Complex = 1+1j
```

```
  File "/tmp/ipykernel_5446/3989137309.py", line 2
    2Complex = 1+1j
     ^
SyntaxError: invalid syntax
```

```python
print('Recmommendation: May not begin with an underscore')
_Grade_Point_Average = 9.41
```

```
Recmommendation: May not begin with an underscore
```

pCPS 2.2.4 keywords and Other Predefined Identifiers in python

- A **keyword** is an identifier that has **predefined** meaning in a programming language.
- keywords **cannot** be used as "**regular**" identifiers.
- There are 35 keywords in python

```python
import keyword
print('keywords in python are\n\n')
KeywordList = keyword.kwlist
print(KeywordList)
print('\n\n The number of keyword present in python are ',len(KeywordList))
```

```
keywords in python are


['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif
', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'o
r', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']


 The number of keyword present in python are  35
```

pCPS 2.2.4 keywords and Other Predefined Identifiers in python

- There are other predefined identifiers that can be used as regular identifiers, but should not be.
- This includes float, int, print, exit, and quit

```
float = 12.0
int = 192
exit=999
quit=1000
print(float)
print(int)
print(exit)
print(quit)
int = float(int)
```

```
12.0
192
999
1000

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_7335/2569839970.py in <module>
      7 print(exit)
      8 print(quit)
----> 9 int = float(int)

TypeError: 'float' object is not callable
```

```
Output = print
print = 100+100j
Output(print)
```

```
(100+100j)
```

```
Float = float
float = 12.0
int = 192
exit=999
quit=1000
print(float)
print(int)
print(exit)
print(quit)
int = Float(int)
```

```
12.0
192
999
1000
```

pCPS 2.2.4 keywords and Other Predefined Identifiers in python

- There are other predefined identifiers that can be used as regular identifiers, but should not be.
- This includes float, int, print, exit, and quit

```
float = 12.0
int = 192
exit=999
quit=1000
print(float)
print(int)
print(exit)
print(quit)
int = float(int)
```

```
12.0
192
999
1000

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_7335/2569839970.py in <module>
      7 print(exit)
      8 print(quit)
----> 9 int = float(int)

TypeError: 'float' object is not callable
```

```
Output = print
print = 100+100j
Output(print)
```

```
(100+100j)
```

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University
nitin.pujari@pes.edu

For Course Digital Deliverables visit www.pesuacademy.com